SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING-IoT Including CS & BCT

COURSE NAME :23ITB201-DATA STRUCTURES & ALGORITHMS

II YEAR / III SEMESTER
Unit III- **SORTING, SEARCHING & HASHING**
Topic :Bubble Sort – Selection Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order.

This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

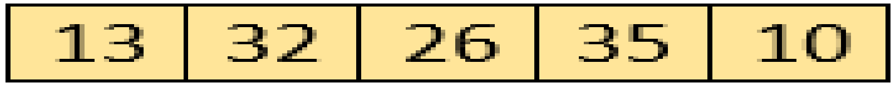
Bubble Sort Algorithm

- ✓ Traverse from left and compare adjacent elements and the higher one is placed at right side.
- ✓ In this way, the largest element is moved to the rightmost end at first.
- ✓ This process is then continued to find the second largest and place it and so on until the data is sorted.

```
begin BubbleSort(arr)
   for all array elements
      if arr[i] > arr[i+1]
         swap(arr[i], arr[i+1])
      end if
   end for
   return arr
end BubbleSort
```

Working of Bubble sort Algorithm

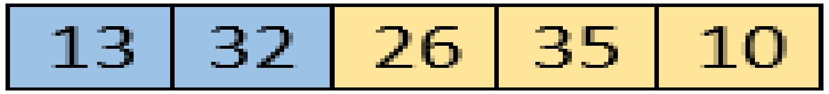
Let the elements of array are -



13	32	26	35	10

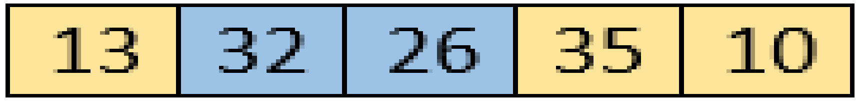
First Pass

Sorting will start from the initial two elements. Let compare them to check which is greater.



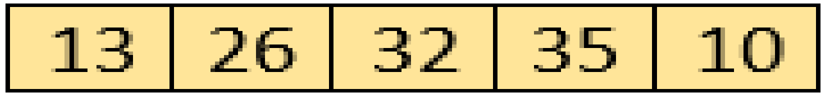
13	32	26	35	10

Here, 32 is greater than 13 (32 > 13), so it is already sorted. Now, compare 32 with 26.



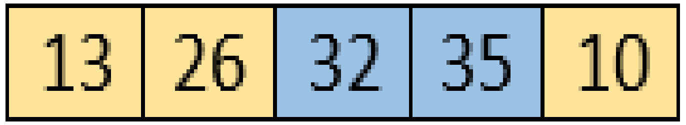
13	32	26	35	10

Here, 26 is smaller than 32. So, swapping is required. After swapping new array will look like -



13	26	32	35	10

Now, compare 32 and 35.



13	26	32	35	10

Here, 35 is greater than 32. So, there is no swapping required as they are already sorted.

Now, the comparison will be in between 35 and 10.

13	26	32	35	10

Here, 10 is smaller than 35 that are not sorted. So, swapping is required.

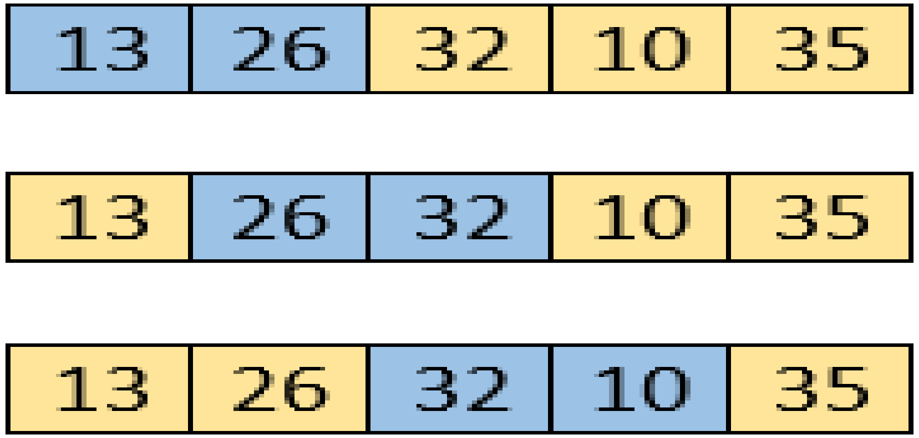
Now, we reach at the end of the array. After first pass, the array will be

13	26	32	10	35

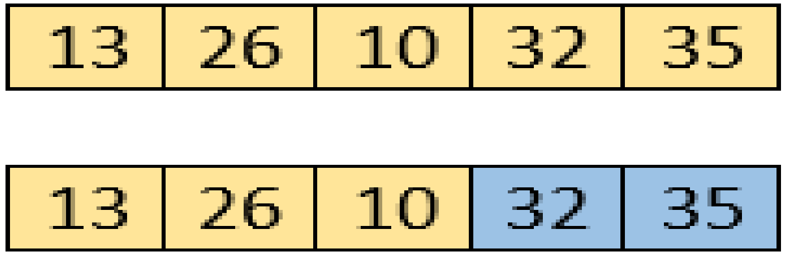
Now, move to the second iteration.

Second Pass

The same process will be followed for second iteration.



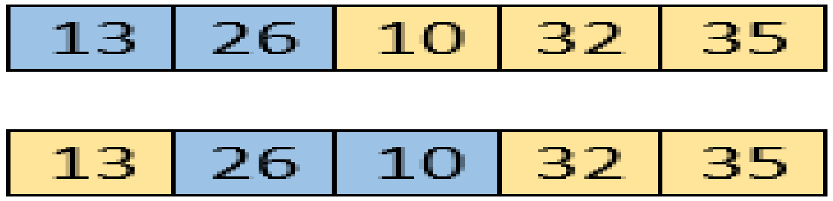
Here, 10 is smaller than 32. So, swapping is required. After swapping, the array will be



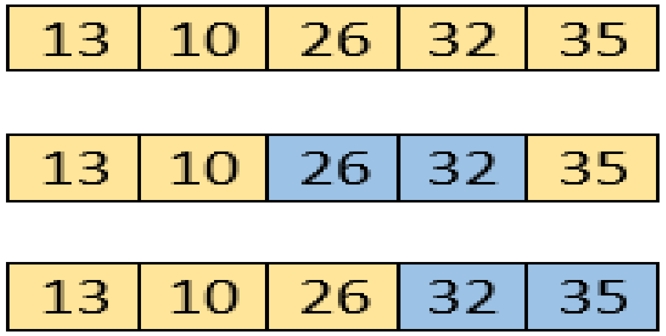
Now, move to the third iteration.

Third Pass

The same process will be followed for third iteration.



Here, 10 is smaller than 26. So, swapping is required. After swapping, the array will be



Now, move to the fourth iteration.

Fourth pass

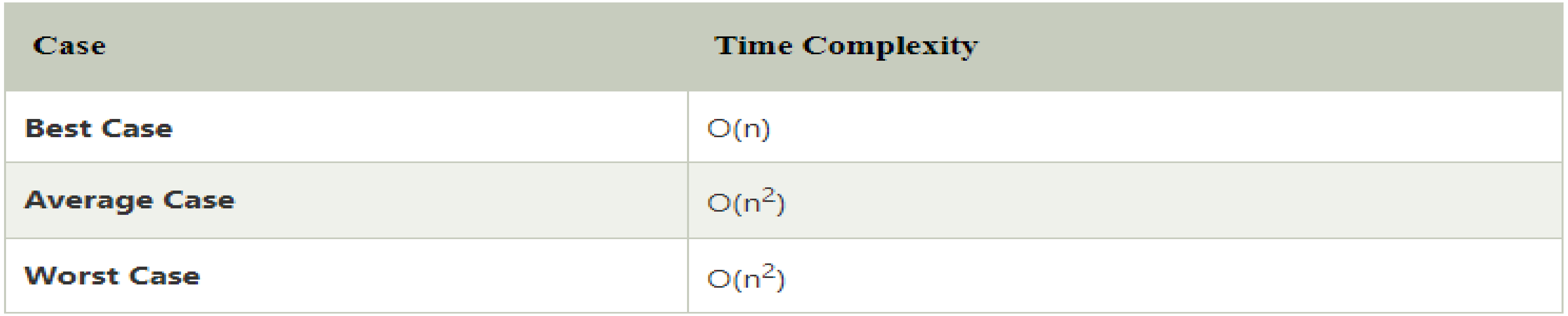
Similarly, after the fourth iteration, the array will be

10	13	26	32	35

Bubble sort complexity

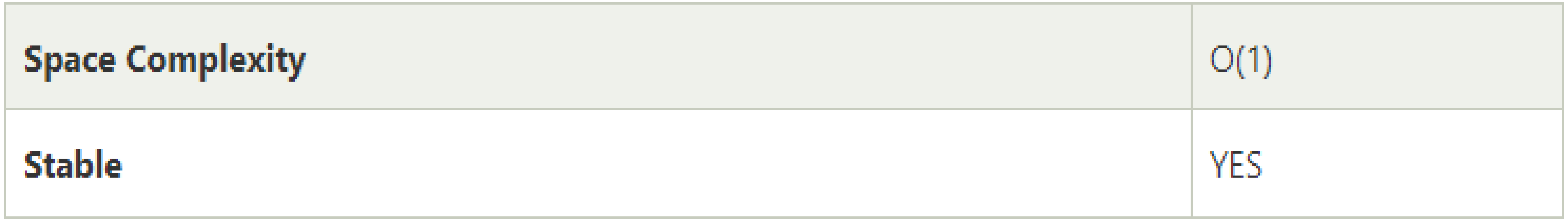
Now, let's see the time complexity of bubble sort in the best case, average case, and worst case. We will also see the space complexity of bubble sort.

1. Time Complexity



Case	Time Complexity
Best Case	O(n)
Average Case	$O(n^2)$
Worst Case	$O(n^2)$

2. Space Complexity



Space Complexity	O(1)
Stable	YES

```
void bubble(int a[], int n) //
function to implement bubble
sort
 {
  int i, j, temp;
  for(i = 0; i < n; i++)
   {
     for(j = i+1; j < n; j++)
      {
         if(a[j] < a[i])
         {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
         }
      }
   }
 }
```

```
void main ()
{
   int i, j,temp;
   int a[5] = { 10, 35, 32, 13, 26};
   int n = sizeof(a)/sizeof(a[0]);
   printf("Before sorting array elements are - \n");
   print(a, n);
   bubble(a, n);
   printf("\nAfter sorting array elements are - \n");
   print(a, n);
}
```

Selection Sort

The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted part.

This process is repeated for the remaining unsorted portion until the entire list is sorted.

Algorithm
SELECTION SORT(arr, n)

```
Step 1: Repeat Steps 2 and 3 for i = 0 to n-1
Step 2: CALL SMALLEST(arr, i, n, pos)
Step 3: SWAP arr[i] with arr[pos]
[END OF LOOP]
Step 4: EXIT

SMALLEST (arr, i, n, pos)
Step 1: [INITIALIZE] SET SMALL = arr[i]
Step 2: [INITIALIZE] SET pos = i
Step 3: Repeat for j = i+1 to n
if (SMALL > arr[j])
    SET SMALL = arr[j]
SET pos = j
[END OF if]
[END OF LOOP]
Step 4: RETURN pos
```

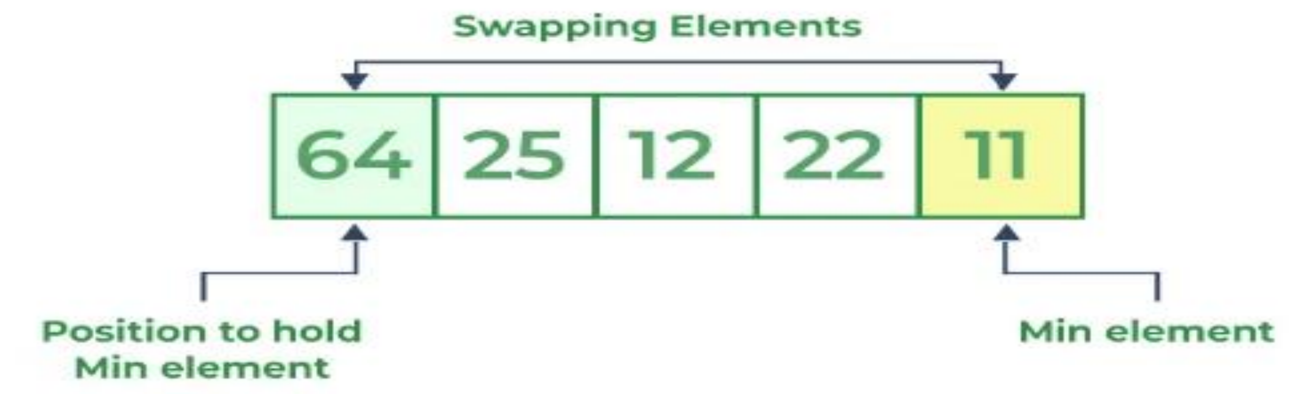
Lets consider the following array as an example:

arr[] = {64, 25, 12, 22, 11}

First pass:

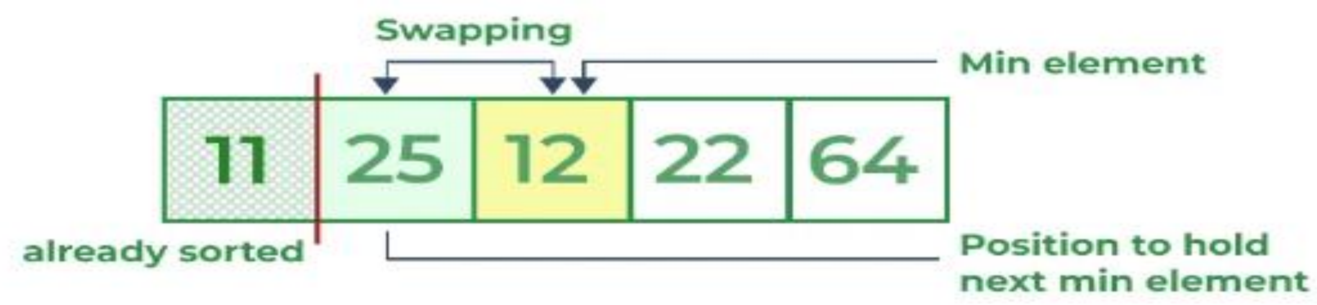
•*For the first position in the sorted array, the whole array is traversed from index 0 to 4 sequentially. The first position where **64** is stored presently, after traversing whole array it is clear that **11** is the lowest value.*

•*Thus, replace 64 with 11. After one iteration **11**, which happens to be the least value in the array, tends to appear in the first position of the sorted list.*


Swapping Elements
64
25
12
22
11
Position to hold Min element
Min element

Second Pass:

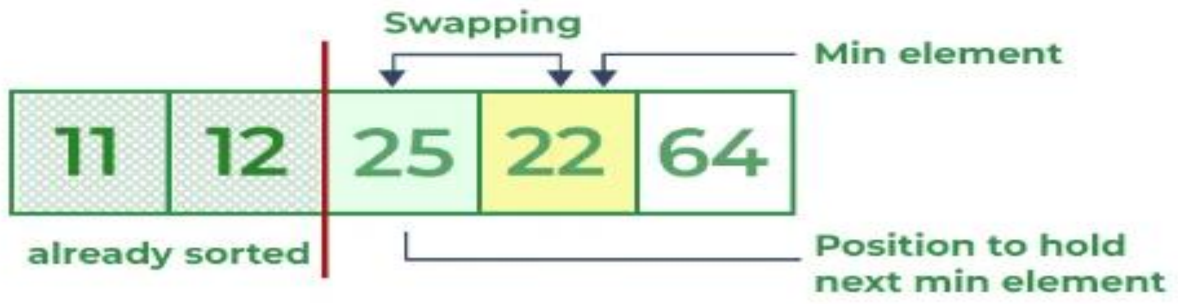
- For the second position, where 25 is present, again traverse the rest of the array in a sequential manner.
- After traversing, we found that **12** is the second lowest value in the array and it should appear at the second place in the array, thus swap these values.



Third Pass:

•Now, for third place, where **25** is present again traverse the rest of the array and find the third least value present in the array.

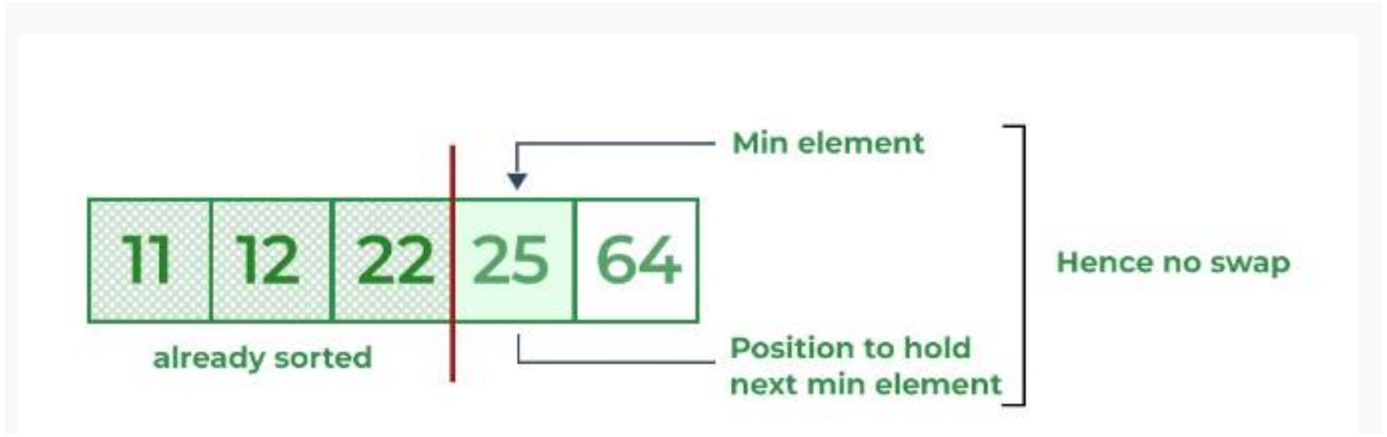
•While traversing, **22** came out to be the third least value and it should appear at the third place in the array, thus swap **22** with element present at third position.



Fourth pass:

•Similarly, for fourth position traverse the rest of the array and find the fourth least element in the array

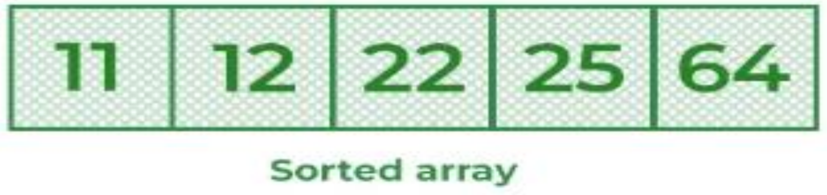
•As **25** is the 4th lowest value hence, it will place at the fourth position.



Fifth Pass:

•At last the largest value present in the array automatically get placed at the last position in the array

•The resulted array is the sorted array.



```
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of
unsorted subarray
    for (i = 0; i < n-1; i++)
    {
// Find the minimum element in unsorted
array
        min_idx = i;
        for (j = i+1; j < n; j++)
          if (arr[j] < arr[min_idx])
            min_idx = j;

        // Swap the found minimum element with
the first element
           if(min_idx != i)
            swap(&arr[min_idx], &arr[i]);
    }
}
```

1. What is the time complexity of Bubble Sort in the worst case?

a) O(n)

b) O(n log n)

c) $O(n^2)$

d) O(log n)

Answer: c) $O(n^2)$

2. In Selection Sort, how many comparisons are made in total to sort an array of size n?

a) n

b) n^2

c) $(n-1)^2$

d) n(n-1)/2

Answer: d) n(n-1)/2

3. Which of the following statements is true regarding Bubble Sort and Selection Sort?

a) Both are stable sorting algorithms

b) Both have the same worst-case time complexity

c) Both are in-place sorting algorithms

d) All of the above

Answer: d) All of the above

4. Which of the following best describes how Bubble Sort works?

a) Repeatedly selects the largest element and moves it to the correct position

b) Repeatedly swaps adjacent elements if they are in the wrong order

c) Selects the smallest element in each pass and moves it to the correct position

d) Divides the array into smaller parts and sorts recursively

Answer: b) Repeatedly swaps adjacent elements if they are in the wrong order

5. What is the number of swaps in Selection Sort to sort an array of size n in the worst case?

a) $O(n^2)$

b) O(n log n)

c) O(n)

d) O(1)

Answer: c) O(n)

Any Query????

Thank you……